Programming Languages and Translators
COMS W4115

Prof. Stephen Edwards
Summer 2008

VINL

VINL Is Not Logo

Language Reference Manual

Yang Cao
June 19, 2008

Table of Contents

1. Introduction

This language manual describes the VINL – VINL is Not Logo – language, which is designed based on the popular C programming language syntactically and Logo programming language functionally.

2. Lexical Conventions

2.1) Tokens

There are six classes of tokens just as in C: identifiers, keywords, constants, strings, operators and other separators. White spaces consists of space, tab and new line, and are separators of tokens.

2.2) Comments

Comments starts with // and terminates with end of line.

2.3) Identifiers

An identifier is a sequence of letters, digits and underscores. Identifiers can not start with a digit. Identifiers are case sensitive.

2.4) Keywords

The following identifiers are reserved for use as keywords, and may not be defined as variable names or function names :

boolean	break	continue	decimal
for	forward	if	pendown
penup	right	void	while
else	print		

2.5) Floating Constants

Floating constants consists of an mandatory integer part, an optional decimal point, and an optional fraction part.

2.6) String

String literals are sequence of characters surrounded by double quotes. A double quote inside the surrounding double quotes is escaped by backward slash '\'. A backward slash inside the double quotes is escaped by backward slash '\' as well.

2.7) Other Tokens

Other tokens used in VINL including: +, -, *, /, %, <-,=, (,), &&, ||, <, >, <=, >=,!=,,

3. Types

VINL is statically typed. All variables must be declared as either a decimal, a string or a boolean. Function declaration must specify a return type. There will be no explicit nor implicit casting in VINL between boolean and decimals. When the concatenation operation is performed on strings and decimal/boolean, the other type will be automatically converted to a string literal. VINL follows Java conversion rule -- a decimal will become a string with digits representing its value and boolean will be either “true” or “false”. The following is a list of the build-in types.

3.1) decimal

A floating constant. The memory storage size of decimal type is not specified. In this implementation, it will be 64bit value. Behaviors on operations on a decimal resulting in overflow , divide check and other exceptions are also not specified.

3.2) String

string of characters.

3.3) boolean

Either true or false

3.4) void

Denotes nothingness. One cannot create a variable of type void. Used only in function declaration to show the function does not return any value.

4. Expressions

4.1) Primary Expressions

Primary expressions include identifiers, constants, strings, function calls, and expressions in parentheses.

a) identifiers

identifiers are lvalues and are evaluated to values stored in this memory space

b) constants

constants are evaluated to itself.

c) strings

strings are evaluated to itself.

d) Function Calls

A function call consists of a identifier declared as a function, followed by optional white space characters, followed by a list of arguments surrounded by "(" and ")".

4.2) Operators

The operators are listed by precedence, highest to lowest, they take primary expressions as operands:

a) Multiplicative Operators

*, /, and % group left to right. Only defined for decimal type constants and identifiers.

b) Additive Operators

+ and – group left to right. - is only defined for decimal type constants and identifiers. + is defined for both string and decimals. When one of the operands is string literal, + is defined as the concatenation operation.

c) Relational Operators

<,>,<=,>= ,!= can be used to compare two decimals, and they yield boolean values. Those operators can not be chained, e.g. (a<b)<c since (boolean)<c is not allowed.

d) Logical AND Operator

&& group left to right and can be used in between two boolean expressions. It yield boolean values. It evaluates to true if and only if both boolean expressions are true.

e) Logical OR Operator

|| group left to right and can be used in between two boolean expressions. It yield boolean values. It evaluates to true if and only if at least one of the operands is true.

4.3) Assignment Expressions

There is only one assignment operator <- and it groups right to left. It requires a type matching identifier on the left side of the operator. The value of the expression on th right of the operator will be stored in the left identifier.

5. Statements

Statements are executed in sequence unless otherwise noted. They do not have values, and are executed for their effects. Multiple statements maybe grouped together by "{" and "}", in that case they are treated as if they are a single statement.

5.1) Expression Statement

Expression statements have the form:

expression-stmt: (expression)? ;

All effects of the expression is completed before the next statement is executed. A empty expression statement is a null statement and is often used for grouping related statements.

5.2) Selection Statement

Selection statements have the form:

selection-stmt : if (expression) statement

| if (expression) statement else statement ;

The "else" groups with the closest else-less if.

5.3) Iteration Statement

Iteration statements have the form:

iteration-stmt : while (expression) statement

| for (expression?;expression?;expression?) statement ;

The while loop stops if and only if the expression evaluates to false. The for loop is equivalent to the following while loop:

```
initialization; //first optional expression
while (expression) //second optional expression
    statement
    post-action //third optional expression
```

The keyword break and continue may only appear in the statement inside a loop body. Break will stop the execution of the inner-most loop immediately; continue will stop the current iteration of the inner-most loop and starts next iteration of this loop immediately.

5.4) Keyword statement

Keyword statements have the form:

keyword-stmt : forward decimal

| right decimal

| print string;

forward and right will impact the turtle, while print will print to the stdout. Print will have no effect on the turtle graph, it should be used as a debugging tool.

6. Scope

All source code can only be stored in one file.

Global variables are variables defined outside of any function body, and can be accessed from everywhere in the source.

Local variables are variables defined inside a function body, and can only be referenced inside the function where it has been declared.